COMP3314_2C_2024 Machine Learning

Programming Assignment 2 (in groups)

Multi-layer perceptron (MLP)

Due date: 11:59pm, 30 April, 2025

Hanzhong Guo

COMP3314_2C_2024 Machine Learning

Programming Assignment 2 (in groups)

Multi-layer perceptron (MLP)

Due date: 11:59pm, 30 April, 2025

Hanzhong Guo

- ✓ The Assignment Requirements
- ✓ An Introduction to PyTorch

✓ The Assignment Requirements

✓ An Introduction to PyTorch

The Requirements

[1] TASK

This assignment is about applying a Multi-layer Perception (MLP) to the task of digit recognition using the PyTorch framework. Students are required to improve the recognition accuracy of this MLP by tuning several hyperparameters and submit a report that summarizes your trials. A code template is provided to facilitate the implementation.

The Requirements (cont)

[2] DATASET

- Street View House Numbers (SVHN) Dataset

This is a 10-category classification problem. The training set contains 3,000 samples for each class while each class in the testing set contains 500 samples. Each input image has three channels (RGB) and each channel contains 32*32 pixels.

We already split the train and test set in the Moodle. Any train on test set won't be allowed

MNIST

SVHN

The Requirements (cont)

[3] GUIDELINES (1)

Students should first learn how to use PyTorch to implement a neural network. An official 60-minute blitz (https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html) will greatly help you to understand the basic components of PyTorch and save you a lot of time. Python is the default programming language and other deep learning frameworks besides PyTorch are prohibited.

The Requirements (cont)

[3] GUIDELINES (2)

In the provided template, you are asked to tune hyper-parameters, including the initial learning rate, decay strategy of the learning rate, and the number of training epochs or even the model structure to improve the performance of the MLP model given in the template.

The Requirements (cont)

Recap of learning rate

Update the weights

$$w_j := w_j + \Delta w_j$$

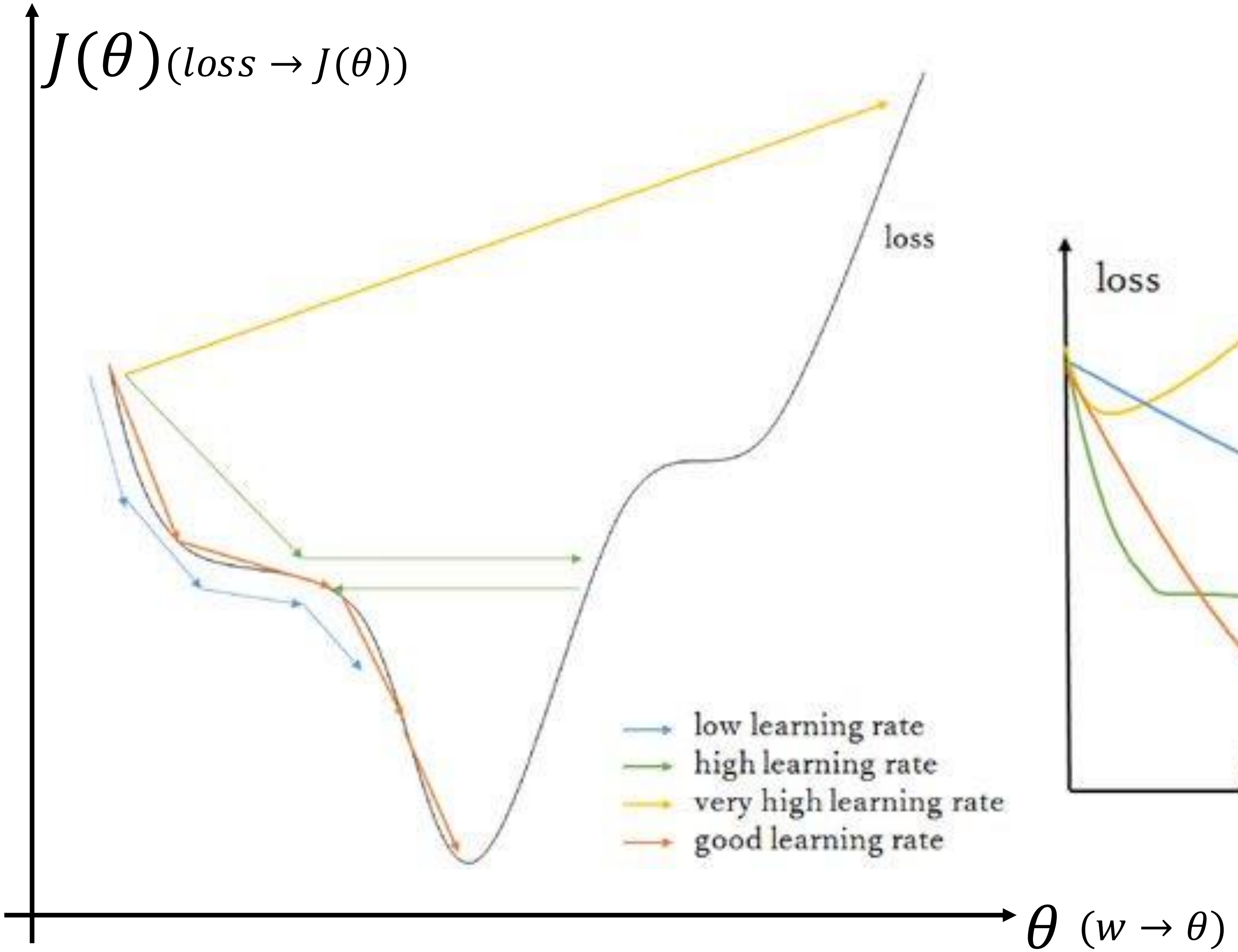
The value of Δw_j is calculated as follows

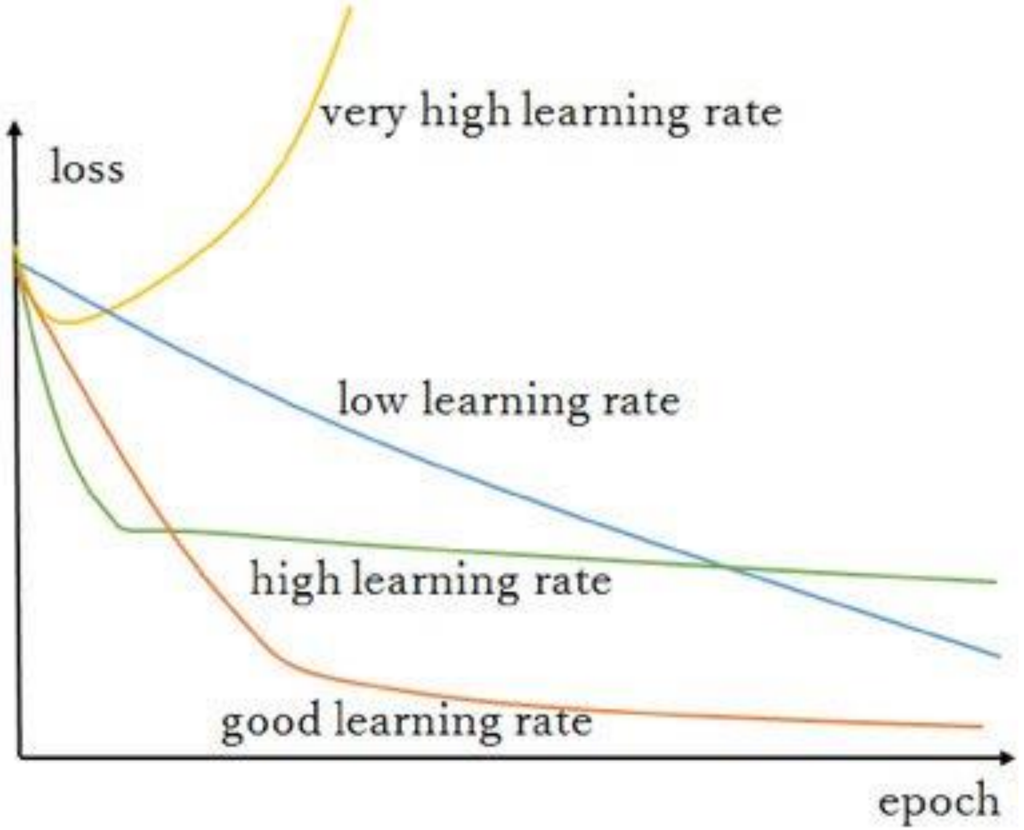
$$\Delta w_j = \boxed{\eta}\left(y^{(i)} - \hat{y}^{(i)}\right)x_j^{(i)}$$

Where η is the learning rate, $y^{(i)}$ is the true class label of the ith training sample, and $\hat{y}^{(i)}$ is the predicted class label

The Requirements (cont)

Initial learning rate

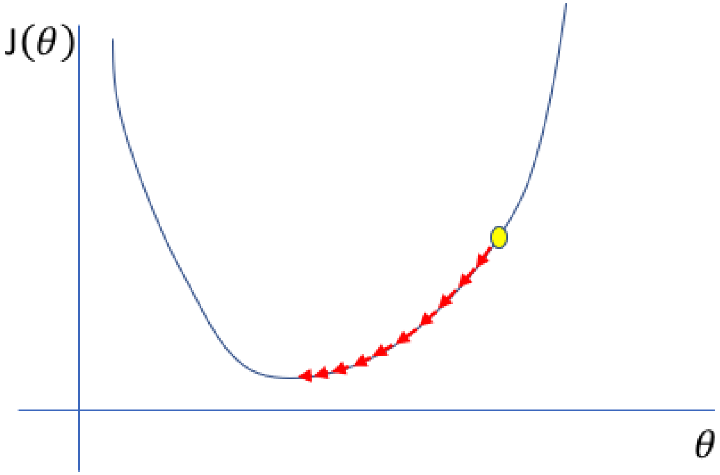




The Requirements (cont)

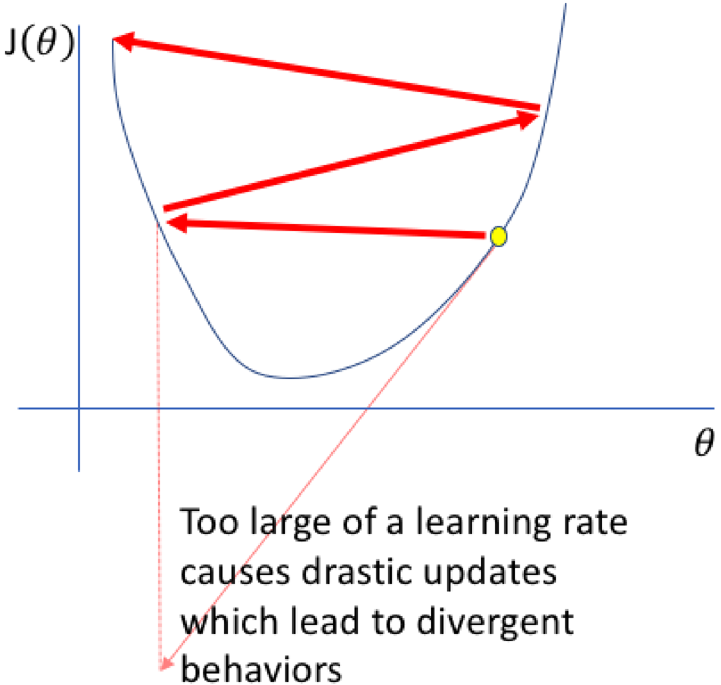
Initial learning rate→ Decay the learning rate

Too low



A small learning rate requires many updates before reaching the minimum point

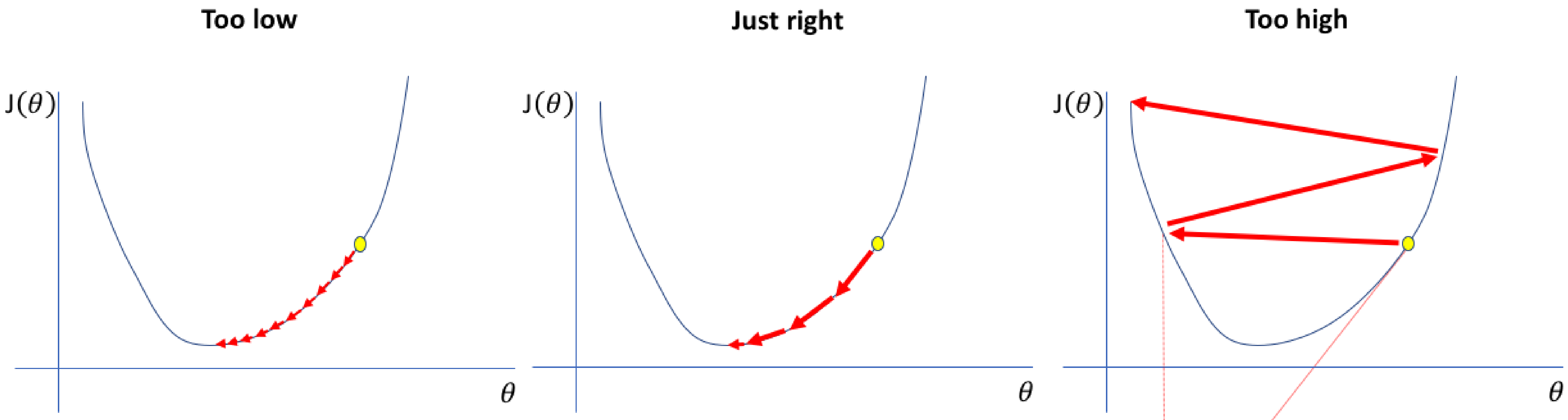
Too high



Too large of a learning rate causes drastic updates which lead to divergent behaviors

The Requirements (cont)

Decay the learning rate → Why?



A small learning rate requires many updates before reaching the minimum point

The optimal learning rate swiftly reaches the minimum point

Too large of a learning rate causes drastic updates which lead to divergent behaviors

✓ Decreasing the learning rate → Local Minimum

✓ A big initial value → Efficient Training

The Requirements (cont)

Decay the learning rate → When?

One of the simplest rules is to decay the learning rate when the loss decreases very slowly (known as plateau).

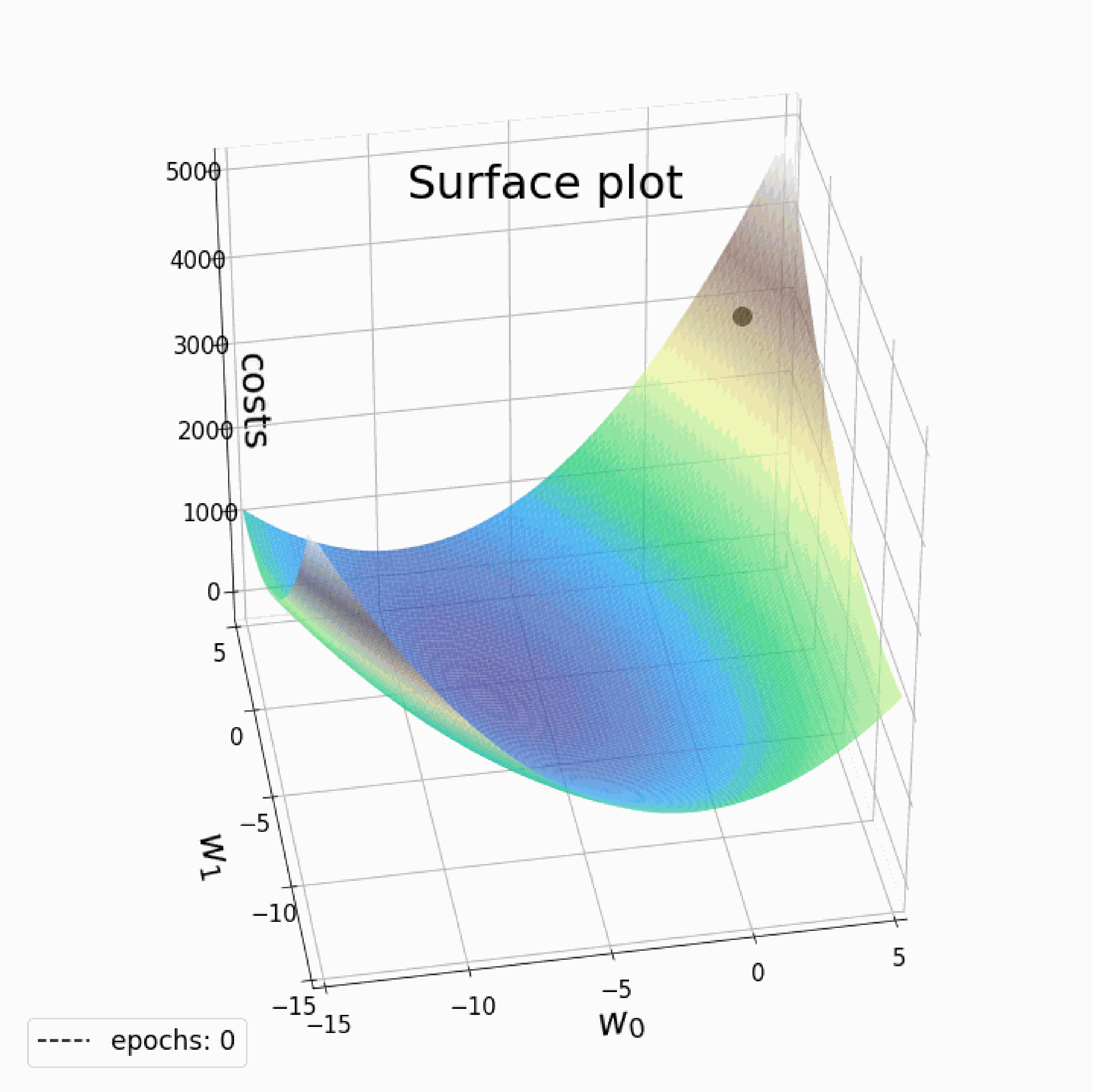
The Requirements (cont)

Number of training epochs

For deep MLPs:

✓Few epochs → Underfitting

✓Too many epochs → Overfitting



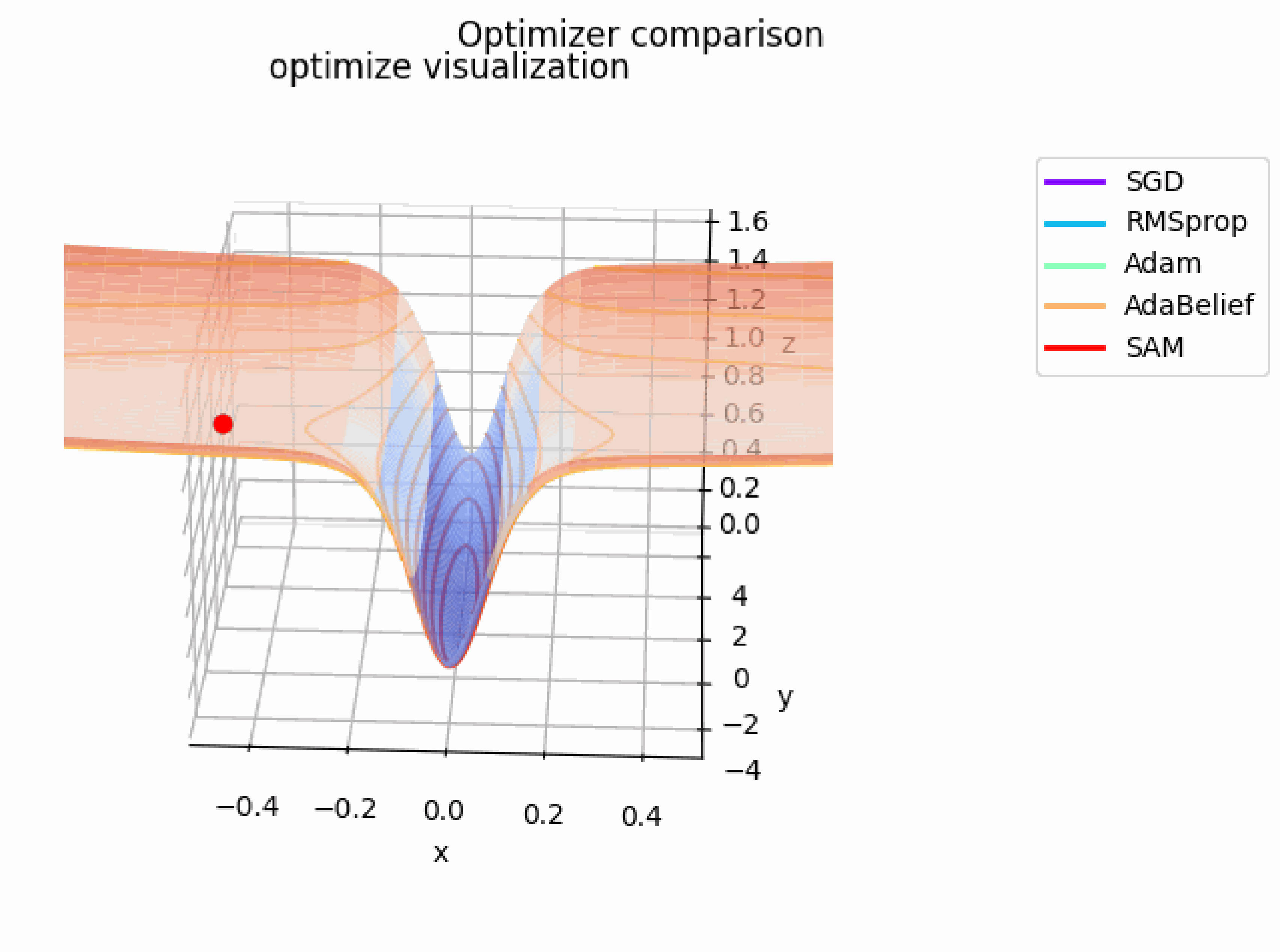
The Requirements (cont)

[3] GUIDELINES (3, Optional)

Besides, you are allowed to change the optimizer (**the default choice is Adam**, an improved version of SGD), data augmentation and network architectures for better training and testing results.

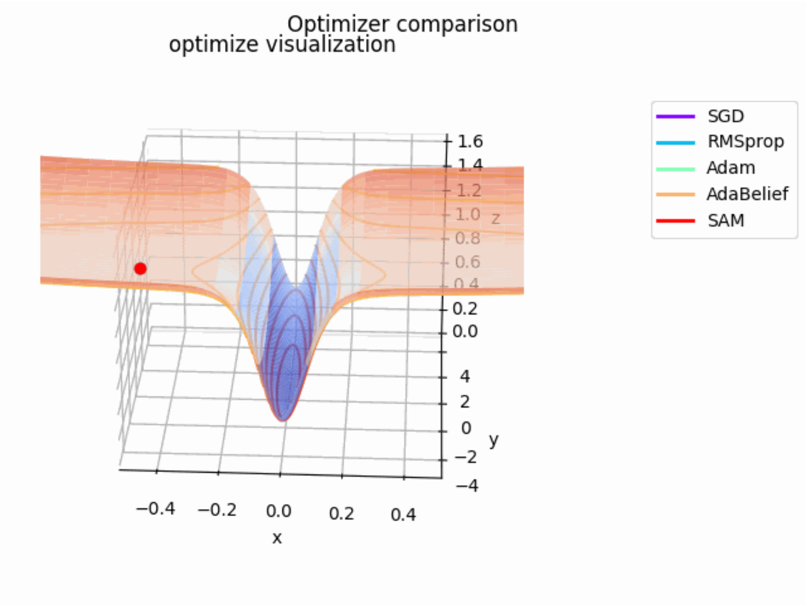
The Requirements (cont)

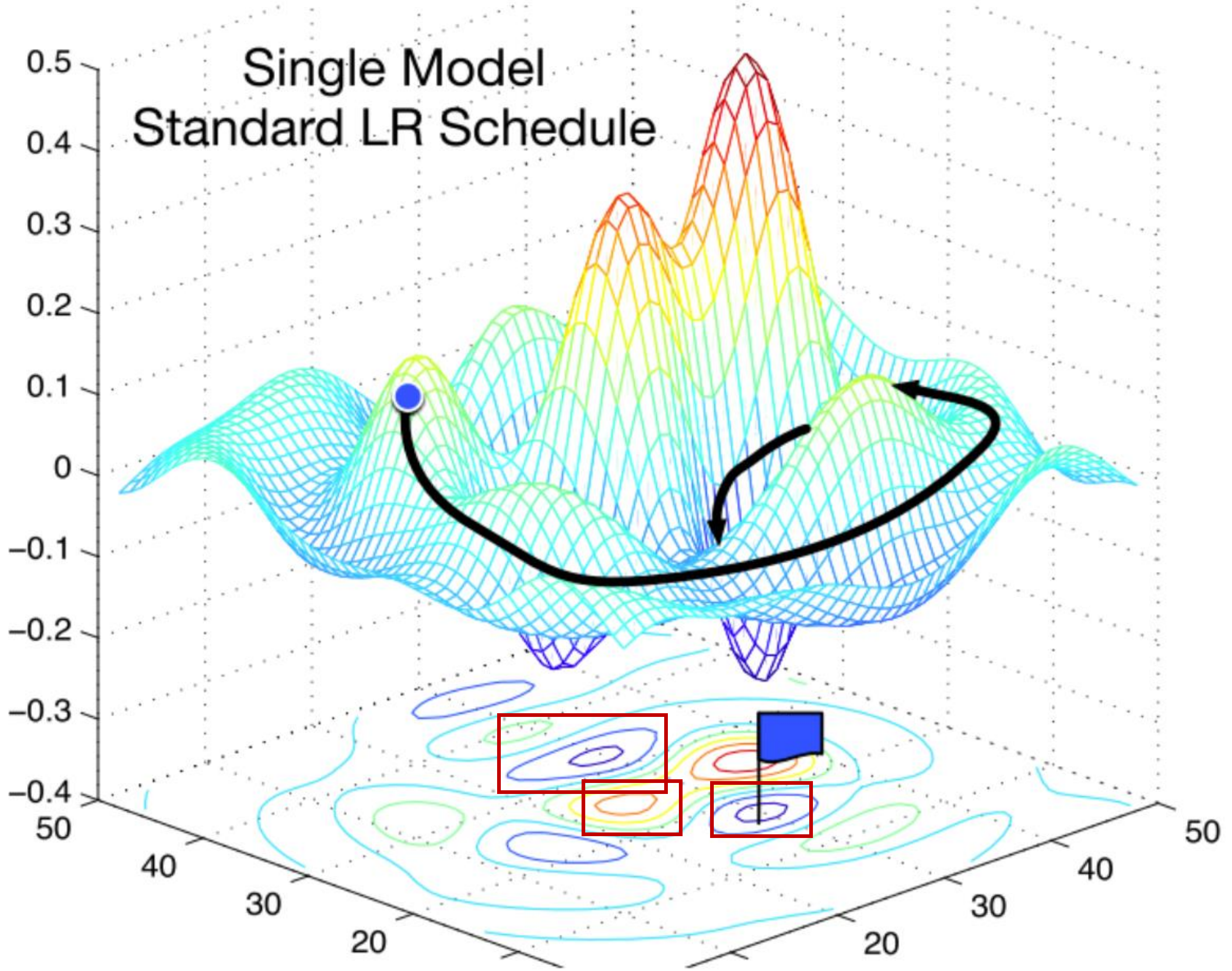
Adam vs. SGD



The Requirements (cont)

Adam vs. SGD





Does not mean Adam is always better than SGD!

The Requirements (cont)

[3] GUIDELINES (4)

Report writing: The report should contain the detailed results of your implementation: overall testing accuracy, accuracy for each class (0-9) and results for different experimental settings. An ideal model should be optimized to achieve an overall accuracy greater than or equal to 80% (the provided template achieves an accuracy of 78%). **We will grade each assignment based on the overall model performance ranking and the report's completeness.**

The Requirements (cont)

[3] GUIDELINES (5)

You should record your computer configuration and the running time of your program in the report. The overall running time should be less than 45 minutes (including training and testing stages). GPU is not recommended in this assignment as the model already runs quite fast on CPUs.

- ✓ The Assignment Requirements
- ✓ An Introduction to PyTorch

An Introduction to PyTorch

PyTorch is among the top 2 choices (the other one is TensorFlow) in deep learning frameworks.

An Introduction to PyTorch (cont)

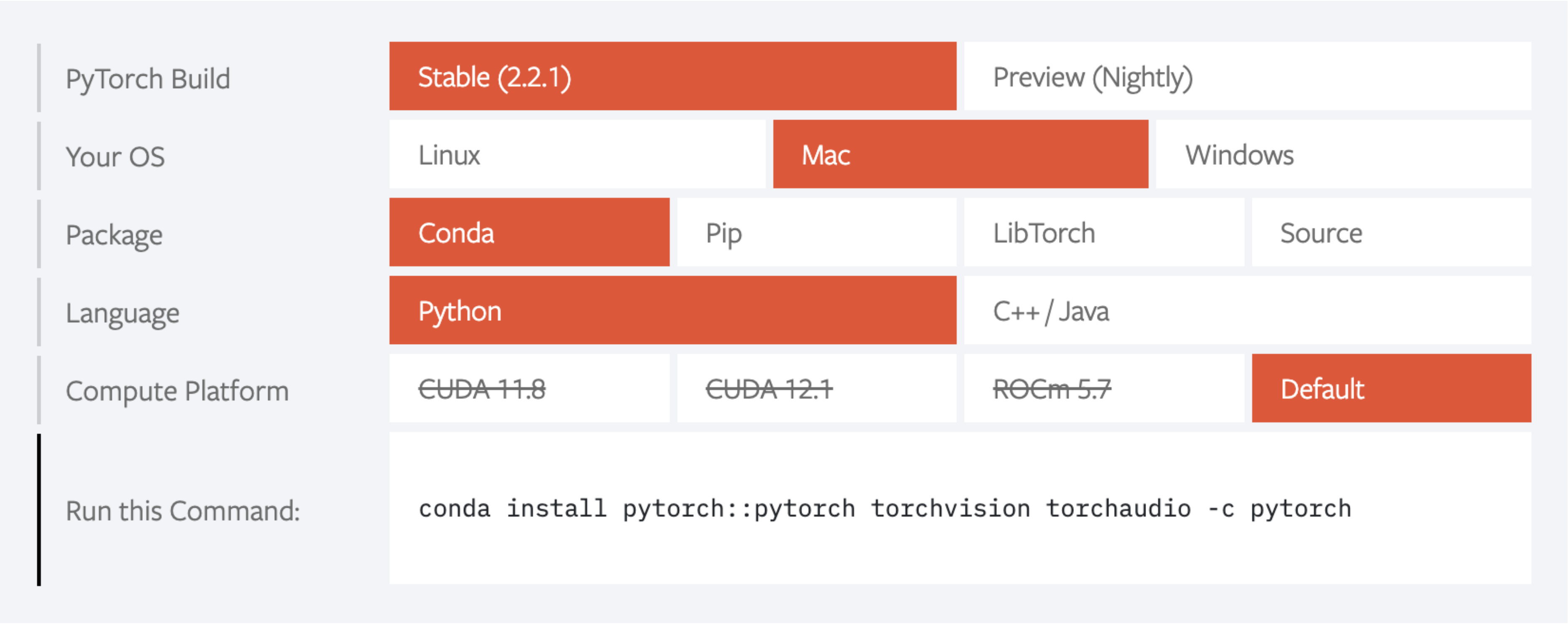
You can quickly build a prototype for MLP using interfaces in PyTorch. More importantly, it is easier to debug in PyTorch than in TensorFlow.

An Introduction to PyTorch (cont)

<u>Install anaconda first!</u>

Installation

Command (example): conda install pytorch::pytorch torchvision torchaudio -c pytorch



You can use other versions of PyTorch. In most cases, the template would run fluently.

https://pytorch.org

An Introduction to PyTorch (cont)

How to define neural networks?

Import python packages

```python
from __future__ import print_function, division

import torch
import torch.optim as optim

from torch.optim import lr_scheduler
from torchvision import datasets, transforms
import time
import os

import torch.nn as nn

class Net(nn.Module):
    """
    Input - 1x32x32
    Output - 10
    """
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(
            nn.Flatten(),  # Flatten the image
            nn.Linear(32 * 32 * 1, 256),  # Fully connected
            nn.ReLU(),
            nn.Linear(256, 512),  # Fully connected
            nn.ReLU(),
            nn.Dropout(0.5),          # Dropout
```

An Introduction to PyTorch (cont)

How to define neural networks?

```python
from __future__ import print_function, division

import torch
import torch.optim as optim

from torch.optim import lr_scheduler
from torchvision import datasets, transforms
import time
import os
```

Import python packages

```python
import torch.nn as nn

class Net(nn.Module):
    """
    Input - 1x32x32
    Output - 10
    """
```

Initialize the class of network

```python
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(
            nn.Flatten(),  # Flatten the image
            nn.Linear(32 * 32 * 1, 256),  # Fully connected
            nn.ReLU(),
            nn.Linear(256, 512),  # Fully connected
            nn.ReLU(),
            nn.Dropout(0.5),            # Dropout
```

An Introduction to PyTorch (cont)

How to define neural networks? (Read the 60-minute blitz)

```
from __future__ import print_function, division

import torch
import torch.optim as optim

from torch.optim import lr_scheduler
from torchvision import datasets, transforms
import time
import os
```

Import python packages

```
import torch.nn as nn

class Net(nn.Module):
    """
    Input - 1x32x32
    Output - 10
    """
```

Initialize the class of network

```
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(
            nn.Flatten(),  # Flatten the image
            nn.Linear(32 * 32 * 1, 256),   # Fully connected
            nn.ReLU(),
            nn.Linear(256, 512),  # Fully connected
            nn.ReLU(),
            nn.Dropout(0.5),            # Dropout
```

nn. Flatten() defines the flatten the 2D images as 1D token

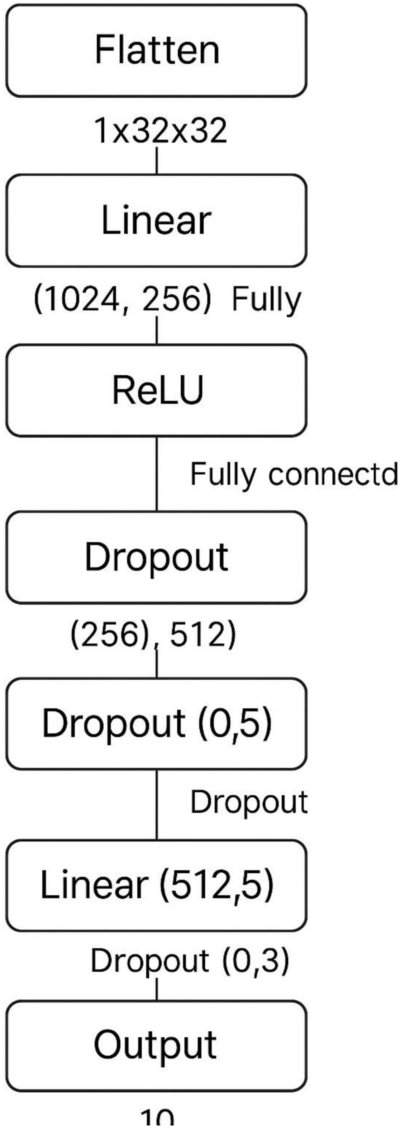
nn.Linear() defines the linear projection

An Introduction to PyTorch (cont)

How to define neural networks?

```python
class Net(nn.Module):
    """
    Input - 1x32x32
    Output - 10
    """
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(
            nn.Flatten(),  # Flatten the image
            nn.Linear(32 * 32 * 1, 256),  # Fully connected
            nn.ReLU(),
            nn.Linear(256, 512),  # Fully connected
            nn.ReLU(),
            nn.Dropout(0.5),          # Dropout
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(256, 10)
        )

    def forward(self, xb):
        return self.network(xb)
```



An Introduction to PyTorch (cont)

Initializing configurations

```
if __name__ == '__main__':
    end = time.time()
    model_ft = Net().to(device) # Model initialization
    print(model_ft.network)
    criterion = nn.CrossEntropyLoss() # Loss function initialization

    # TODO: Adjust the following hyper-parameters: initial learning rate, decay strategy of the learning rate,
    #       number of training epochs
    optimizer_ft = optim.Adam(model_ft.parameters(), lr=1e-3) # The initial learning rate is 1e-3

    # Decay strategy of the learning rate
    exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=10, gamma=0.7)

    history, accuracy = train_test(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                 num_epochs=15) # The number of training epochs is 15
```

An Introduction to PyTorch (cont)

Initializing configurations

```
if __name__ == '__main__':
    end = time.time()
    model_ft = Net().to(device) # Model initialization
    print(model_ft.network)
    criterion = nn.CrossEntropyLoss() # Loss function initialization

    # TODO: Adjust the following hyper-parameters: initial learning rate, decay strategy of the learning rate,
    #       number of training epochs
    optimizer_ft = optim.Adam(model_ft.parameters(), lr=1e-3) # The initial learning rate is 1e-3

    # Decay strategy of the learning rate
    exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=10, gamma=0.7)

    history, accuracy = train_test(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                num_epochs=15) # The number of training epochs is 15
```

An Introduction to PyTorch (cont)

Initializing configurations

```
if __name__ == '__main__':
    end = time.time()
    model_ft = Net().to(device) # Model initialization
    print(model_ft.network)
    criterion = nn.CrossEntropyLoss() # Loss function initialization

    # TODO: Adjust the following hyper-parameters: initial learning rate, decay strategy of the learning rate,
    #       number of training epochs
    optimizer_ft = optim.Adam(model_ft.parameters(), lr=1e-3) # The initial learning rate is 1e-3

    # Decay strategy of the learning rate
    exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=10, gamma=0.7)

    history, accuracy = train_test(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                num_epochs=15) # The number of training epochs is 15
```

An Introduction to PyTorch (cont)

Initializing configurations

```
if __name__ == '__main__':
    end = time.time()
    model_ft = Net().to(device) # Model initialization
    print(model_ft.network)
    criterion = nn.CrossEntropyLoss() # Loss function initialization

    # TODO: Adjust the following hyper-parameters: initial learning rate, decay strategy of the learning rate,
    #       number of training epochs
    optimizer_ft = optim.Adam(model_ft.parameters(), lr=1e-3) # The initial learning rate is 1e-3

    # Decay strategy of the learning rate
    exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=10, gamma=0.7)

    history, accuracy = train_test(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                num_epochs=15) # The number of training epochs is 15
```

LR= 0.001 LR= 0.0007

1-10 epochs 11-15 epochs

An Introduction to PyTorch (cont)

A simple training procedure

```
for i, data in enumerate(train_dataloader, 0):
    inputs, labels = data[0].to(device), data[1].to(device)
    optimizer.zero_grad()
    outputs = model(inputs)        Forward inputs
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
```

An Introduction to PyTorch (cont)

A simple training procedure

```
for i, data in enumerate(train_dataloader, 0):
    inputs, labels = data[0].to(device), data[1].to(device)
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
```

Compute loss values, criterion is a predefined loss function

An Introduction to PyTorch (cont)

A simple training procedure

```
for i, data in enumerate(train_dataloader, 0):
    inputs, labels = data[0].to(device), data[1].to(device)
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
```

Backward automatically

An Introduction to PyTorch (cont)

A simple training procedure

```
for i, data in enumerate(train_dataloader, 0):
    inputs, labels = data[0].to(device), data[1].to(device)
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
```

Updating the model

An Introduction to PyTorch (cont)

How to improve via the network? (Bonus)

```python
class Net(nn.Module):
    """
    Input - 1x32x32
    Output - 10
    """
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(
            nn.Flatten(),  # Flatten the image
            nn.Linear(32 * 32 * 1, 256),  # Fully connected
            nn.ReLU(),
            nn.Linear(256, 512),  # Fully connected
            nn.ReLU(),
            nn.Dropout(0.5),          # Dropout
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(256, 10)
        )

    def forward(self, xb):
        return self.network(xb)
```

1. We now use the gray images (one channel), we can use the RGB images (three channel)

   Change the dataloader and network
2. A more deeper network will be more useful

Thanks

Email: comp3314.forta@gmail.com

My email: hanzhong@connect.hku.hk